282. Expression Add Operators

HARD

Topics

- Math
- String
- Backtracking

Problem description: Given a string num that contains only digits and an integer target, return ***all possibilities*** *to insert the binary operators* '+', '-', *and/or* '*' *between the digits of* num *so that the resultant expression evaluates to the* target *value*.

The expression should not contain leading zeros, and a number can contain multiple digits. Example having “09” is not allowed.

The multiply operand has higher precedence than addition and subtraction.

Example:

1 + 2 - 4 * 12 --> -1 * 12 --> -12 (WRONG!)

1 + 2 - 4 * 12 --> -1 - (-4) + (-4 * 12) --> 3 + (-48) --> -45 (CORRECT!)

By Nickolas Diaz

Examples

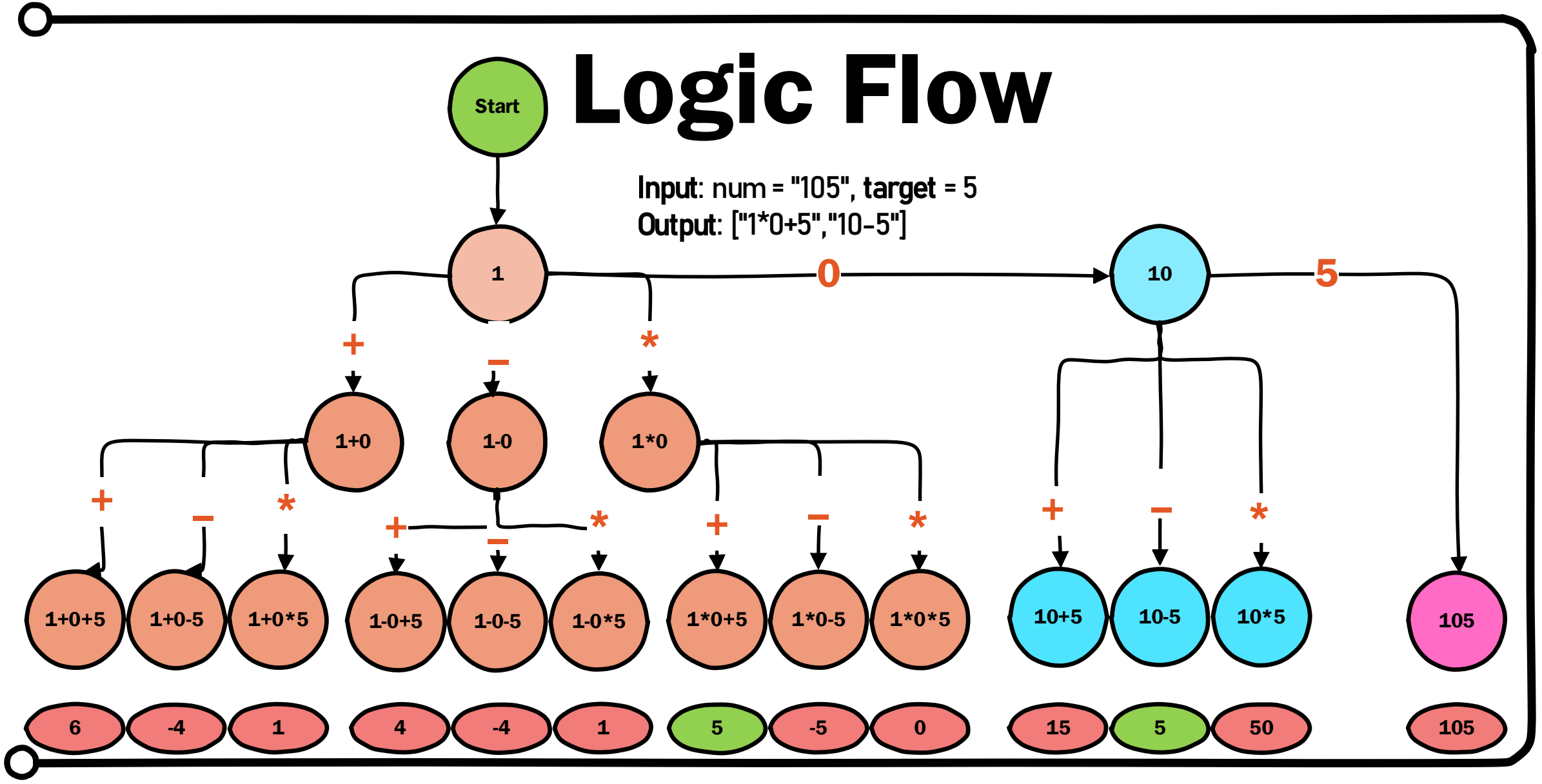
Example 1

- **Input:** num = "123", target = 6
- **Output:** ["1*2*3","1+2+3"]
- **Explanation:** Both "1*2*3" and "1+2+3" evaluate to 6.

Example 2

- **Input:** num = "232", target = 8
- **Output:** ["2*3+2","2+3*2"]
- **Explanation:** Both "2*3+2" and "2+3*2" evaluate to 8.

Logic Flow


Start
Input: num = "105", target = 5
Output: ["1*0+5","10-5"]
1
0
10
5
+
-
*
1+0
1-0
1*0
1+0+5
1+0-5
1+0*5
1-0+5
1-0-5
1-0*5
1*0+5
1*0-5
1*0*5
10+5
10-5
10*5
105
6
-4
1
4
-4
1
5
-5
0
15
5
50
105

Solution Approach

- Using dfs: Taking input of list of numbers and operands, the index of the string and the evaluation of that current state and a string of the current operand.
- Call recursion three times, appending “*”, “-”, and “+”.
- Example: 1+0, 1-0, 1*0
- Call recursion for each digit, Example: 10, make sure to avoid trailing zeros, so 01 is not allowed.
- Once the index, reaches end of the string, compare calculate the string to get its value.

```
class Solution {
    public:
        long calc_str(string expr, vector<long> &nums){

            int x = 1;
            int i = 0;
            long total = nums[0];
            long temp = 0;
            bool temp_assigned = false;
            while(i < expr.size()){
                long val = nums[x];
                switch(expr[i]){
                    case '+':{
                        if(temp_assigned)
                            total += temp;
                        temp = val;
                        temp_assigned = true;
                        break;
                    }
                    case '-':{
                        if(temp_assigned)
                            total += temp;
                        temp = -val;
                        temp_assigned = true;
                        break;
                    }
                    case '*':{
                        if (!temp_assigned){
                            total *= val;
                        }
                        else
                            temp *= val;
                    }
                }
                i++;
                x++;
            }
            return total + temp;

        }
        string op_str(string expr, vector<long> &nums){

            string result {};
            for(int i = 0, x = 0; i < expr.size(); i++, x +=2){
                result += to_string(nums[i]);
                result.push_back(expr[i]);
            }
            result += to_string(nums[nums.size()-1]);
            return result;

        }
```

```
void dfs(vector<long> nums, string cmd, int i, vector<string>& result,
         string& num, int target){
            if(i < num.size()){
                if(!nums.empty() && nums[nums.size()-1] == 0){
                    nums.push_back(num[i] - '0');
                }
                else if(i >= 1){
                    long temp = nums[nums.size()-1];
                    nums[nums.size()-1] *= 10;
                    nums[nums.size()-1] += num[i]-'0';
                    dfs(nums, cmd, i+1, result, num, target);

                    nums[nums.size()-1] = temp;
                    nums.push_back(num[i] - '0');

                } else{
                    nums.push_back(num[i] - '0');
                    dfs(nums, cmd, i+1, result, num, target);
                    return;
                }
            }

            if(i < num.size()){
                i++;
                dfs(nums, cmd + "*", i, result, num, target);
                dfs(nums, cmd + "-", i, result, num, target);
                dfs(nums, cmd + "+", i, result, num, target);
            }
            else{
                if(calc_str(cmd, nums) == target){
                    result.push_back(op_str(cmd, nums));
                }
                return;
            }

        }


        vector<string> addOperators(string num, int target) {
            int n = num.size();
            vector<string> result {};

            dfs({}, {}, 0, result, num, target);

            return result;
        }

    };
```

Solution

TOO SLOW!!!

Time Limit Exceeded 17 / 25 testcases passed

submitted at Oct 31, 2025 10:59

Editorial

Last Executed Input

Use Testcase

num =

"2147483648"

target =

-2147483648

Optimized Solution Approach

- This is like the previous solution but instead of calculating the string expression when comparing it to the result. The expression is calculated on the fly.
- The Logic for Handling Multiplication:
- Subtracting the last operand from the current value (effectively removing the last addition/subtraction).
- Multiplying the last operand by the current number to get the new, higher-precedence value.
- Adding this new product back to the current value.

```cpp
class Solution {
public:
    vector<string> addOperators(string num, int target) {
        vector<string> result;

        dfs(num, target, {}, {}, {}, {}, result);
        return result;
    }

private:
    void dfs(const string& num, int target, int pos, long long eval, long long multed,
             string path, vector<string>& result) {

        if (pos == num.length()) {
            if (eval == target) {
                result.push_back(path);
            }
            return;
        }

        for (int i = pos; i < num.length(); i++) {
            string cur = num.substr(pos, i - pos + 1);

            if (cur.length() > 1 && cur[0] == '0') {
                break;
            }

            long long curVal = stoll(cur);

            if (pos == 0) {
                dfs(num, target, i + 1, curVal, curVal, cur, result);
            } else {
                dfs(num, target, i + 1, eval + curVal, curVal,
                    path + "+" + cur, result);

                dfs(num, target, i + 1, eval - curVal, -curVal,
                    path + "-" + cur, result);

                dfs(num, target, i + 1, eval - multed + multed * curVal,
                    multed * curVal, path + "*" + cur, result);
            }
        }
    }
};
```

Optimized Solution

Arguments for dfs:
Int pos: the index of input string
Long long eval: it is the number of the equation for that state
Long long multed: helper in order to calculate the operations of the
Why long long: because of integer overflow errors
String path: the string operation at that state

14-19: checks when the pos is at the end, if so check the eval is equal to the target
24-26: essentially stops any leading zero
30-31: checks if pos == 0, calls dfs: with the loop, makes sure all digits are executed: 1 -> 10 -> 105
32-41: calls all dfs diversions from ‘+’, ‘-’, ‘*’.
39-41: sets eval to eval – multed + (multed * curVal) and sets multed to multed * curVal
Hanles cases like 1 + 2 * 3, where 1 + 2 happens first and the result is 3* 3 instead of 1 + 6.

Accepted 25 / 25 testcases passed

Nickolas Diaz submitted at Oct 31, 2025 17:34

Editorial | Solution

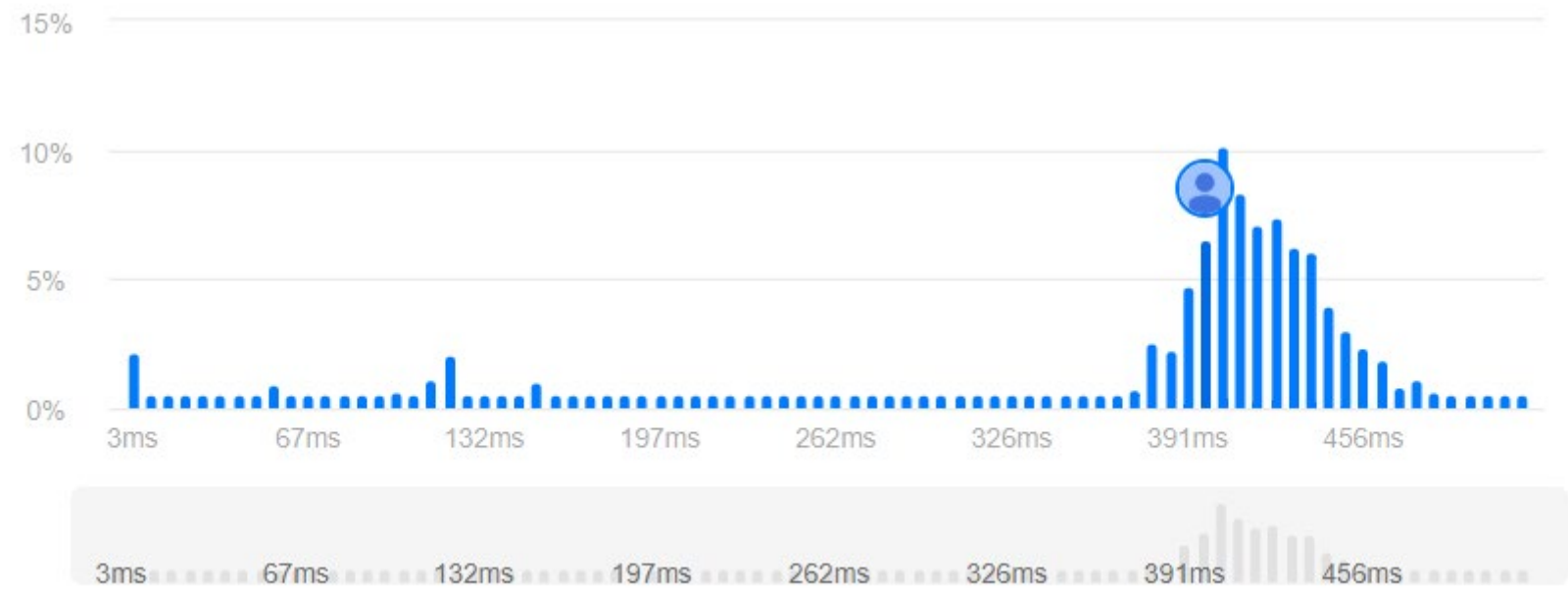
Runtime

403 ms | Beats **66.14%**

Analyze Complexity

Memory

93.26 MB | Beats 55.31%



Time/Space Complexity

- For the unoptimized solution, the time complexity is $O(4^N * N)$. This is because each operand +, -, and * create new branches per node, in addition to each new digit creating a new branch. The N represents calculating the expression from the string for every branch.
- For the optimized on the fly solution, the time complexity is $O(4^N)$. It has the same number of branches as the previous solution but does not need to illiterate through a string for every branch.
- The space complexity for both solutions are O(log(N)). One new node is needed for the depth that it goes.

References

- https://leetcode.com/problems/expression-add-operators/description/?envType=problem-list-v2&envId=backtracking
- https://algo.monster/liteproblems/282
- https://leetcode.com/problems/expression-add-operators/solutions/7324683/c-unoptimized-and-on-the-fly-solution-pr-s5hg/